We saw that in basic authentication the client will have to explicitly keep

adding in the authorization field containing the username and

password for every request that the client sends to the server side.

Now that is perfectly fine for simple authentication.

Cookies are yet another mechanism that is provided that enables your server

to be able to expect the client to store some information on the client side and

include that information explicitly in each outgoing request.

So instead of including your base-64 encoded username and

password like we did in the basic authentication, using cookies,

your server may set up an explicit piece of information on the client side

which then will be included in each outgoing request from the client side.

If your server wants to track information about your

client, then the server may set up explicitly a session tracking mechanism.

Now, cookies are small and can't store a lot of information in there,

and this of course can not be included in the outgoing request.

Cookies can include some basic information

in the header of the outgoing request from the client.

Now, if we want a lot more of information to be tracked about a client

on the server side, then express-sessions enable us to do that.

What are COOKIES ?

HTTP cookies, as mentioned, are small piece of data that is sent

from a web server and is stored on the client side.

Now, almost all browsers have the ability to support

the storing of cookies on the client side, and automatically including them

in the request when the request is sent to a specific server.

So each subsequent request from the client

side will include a new header in there

with the cookie in the request header.

Now how does this cookie setting inclusion in the outgoing request work?

When a client sends a request to the server site,

if the client is authenticated on the server site.

For example, using the basic authentication,

then the server may in return, set up a cookie.

Now, to set up a cookie on the client's site, the server will include in

the response message a header with the sent cookie header and

the actual cookie in the header.

Now when the client receives the response message from the server containing

the Set-Cookie header, then it'll set up the cookie on the client side.

Such that each subsequent request going from the client side will

explicitly include a header field called as cookie and

actual header that contains as the value,

the cookie information that has been sent by the server in the response message.

So each subsequent request message will carry this cookie in the header.

Thereby when the server receives this request message it is able to examine

the cookie and surmise who this request is coming from.

So it is able to recognize the client by looking at the cookie information.

So this is where cookies prove very useful in

being able to send authorization information.

So in serving including username and password as part of the basic

authentication header in every ongoing request.

The first time you authenticate yourself, you send your username and password and

the server sets up the cookie on your side.

Subsequently, you only need to include the cookie in the outgoing request.

Now cookies also can have an expiry date associated with them.

So thereby, at that point, the cookie will be deemed as expired and

will no longer be valid.

So that is one way of controlling the duration for

which an authorization is valid

EXPRESS SUPPORTS COOKIES ?

Express uses a lot of middleware.

This is where one of the middlewares that comes in called the cookie parser

comes to our app at eight.

The cookie-parser allows the server to set up a cookie in the response header.

So this is done by using res.cookie and the name and

certain values

And so cookies, when they are sent from the client-side, included in that request message are parsed on the Express server side, using the cookie-parser.The cookie-parser middleware,

which when installed will enable you to parse the incoming cookies.

And then these incoming cookies will be added into

the request as a header and can be examined on the server side.

Now in order to protect the authenticity of the cookie,

the cookies themselves can be signed by the server.

Now when the server signs a cookie, the server uses a secret key,

which is only known to the server side.

a secret key is

a specific string that only the server knows and nobody else knows.

So when a server encrypts a cookie, it will use a secret key as a signature and

create what is called as a key-hash message authentication code.

And includes this in that cookie that is sent from the server to the client side.

This HMAC that is created on the server side can only be done

by that specific server knowing that secret key.

Now, since the server is a protected resource, so

only the server will know the secret key and so it is very easy to verify

when a signed cookie is sent from the client side to the server side.

So when the signed cookie is sent from the client side to the server side,

the cookie will be set up on the client side, and

then all subsequent requests will include this signed cookie in the client side.

Now the cookie parser middleware that we set up with our Express server

already supports signed cookies

Now for this, in the cookie parser, you will also supply the secret key as

the parameter for the cookie parser when you set up the cookie parser middleware.

Thereby, all the cookies will be signed appropriately and then sent out.

And when the cookie is parsed on the server side in the incoming

request message, this will be added into the request message as req.signedCookies.

And then you can have a specific field which you can check in the signed cookie.

But of course, cookies have limitations.

They are a fixed size, so they cannot encode a lot of information

about the client that their server can retrieve from the cookie.

The cookie is used to just remind the server about

which client is sending the request.

SESSIONS

Now if you want to have a more elaborate mechanism to track information about

a client, then on the server side you can set up what are called as sessions

Now, sessions is a generic mechanism that is available with any servers.

In particular, we'll look at Express itself and

how Express supports session management on the server side.

The way it works is that the user session is set up on the server side.

session itself is a combination of a cookie and

a session ID, and the server-side tracks information

associated with that session ID, or indexed by that session ID.

The session information itself can have any

amount of information being tracked on the server-side, and indexed by that cookie.

So when a client sends a request over the server, then from within the cookie

the session ID is retrieved and that is used as an index into the server side.

For example,

if you are using a server side database that index will be the primary index into

that particular server side database which tracks the sessions.

And thereby, additional information about that session can be retrieved and

used by your server in order to make decisions on how

it services the incoming client request. Now, by default, the sessions are stored in memory on the server site.

Now obviously, what this means is that if your server is restarted, your memory

will be cleared and so all the session information will be gone completely.

So instead, many servers will resort to

using some form of permanent storage where the session information is tracked.

The permanent storage on the server side could either be done

through some kind of a file storage.

Or even leverage the fact that you already have a database on the server side and

than store the session information on the server side.

For example, in your MongoDB itself.

Now in the exercise that follows, we will look at the use of a file storage for

tracking session information on the server side.

Another aspect that you need to pay attention to is the fact that if you

are having a distributed server implementation whereby multiple

servers are acting as the server for servicing the request.

Then the distributor server should be able to access the session information.

Any one of this servers should be able to access the session information.

So you will need a distributor sessions tool on the server side,

to enable you to support multiple replicated servers.

Especially this is useful when we are trying to ensure reliability

of server operation.

Express uses the express-session middleware that supports

the use of sessions in an Express server.

And in the exercise, I'm going to use the FileStore

as a way of permanently storing the session information.

And so we will also include the session-file-store node module that

enables us to use the files on the server side to track the session information.

And then once we do that your session itself will be set up on with the new

express server by declaring the middleware here as session which takes

a certain set of options as a parameter here.

The options include the name for the session, so we'll give the session-id for

the particular session.

And then you'll also supply the secret, a secret key that is used for

encoding the signed cookie that'll be sent to the client side.

And then also additional information including saveUninitialized, which

will be a flag that is used and also a resave flag that is used.

IMPORTANT

And if you are FileStore as the permanent storage for your sessions,

then we will declare that also in the session options there.

So this is how we would set up a session on the express server side

using the express session Middleware.

And the express-session Middleware, when the client sends this information,

this will be parsed on the server side and

this will result in a property called a session being added to the request object.

So this session information will be accessible in the request

object as req.session.

So the req.session will carry additional information about that particular session

for that particular client.

Now once this session, incoming request is parsed by the session middleware,

the req.session property will be added to the incoming

request message object that express uses.

So after the session is parsed, direct session property will be available and

we can examine that too to check which client has sent this request.

When they setup their session object on server site,

as we saw, we can setup various options for that server site.

The cookie, the options from the session ID cookie and the default value for

the cookie will be as shown here, which is path: '/',

httpOnly: true, secure: false, maxAge: null.

So this will be the default value of the cookie that will be stored on the package

and sent over to the client's side as a signed cookie.

And this would be included in every incoming request from the client's site.

Then the genid is the function that generates the session ID.

The default is to use the UUID of the server itself as the general ID.

Then the resave flag, if it is true, forces a session to be saved back to

the store even if it's not modified by the request.

Sometimes the incoming request may

15:45

contain a need to modify the session information on the server side.

And so, if the session information is modified, it'll have to be persistent.

If not, then you don't need to persist it.

But if you set the resave flag to true, even if the session information on

the server Is not modified by the incoming client request, it'll still be resaved.

The next flag that we looked at was saveUninitialized.

If this is true, it'll create a newly created session without any modifications

to be saved of the session store.

Now we will set this to false by default, which means that we only will

track those sessions that are authorized on the server.

secret key that is used for signing the cookie,

and the store itself specifies the session store instance that is used.

The default is to use the in memory store.

You can specify the file store or Mongo store for

storing that session information, and so on.

So once you specify this information for your express session middleware then

the session will be appropriately set up and so will be tracked on the server side.

Each client request will then be mapped to the session information on

the server side when the client request is parsed by the express session middleware.

And the req.session will be added into the request object.